AxiCat

Multipurpose I/O Card

User Manual

December 2014

Table of Contents

1 Features **4**

2 Technical Specifications **5**

3 AxiCat **6**
- Board Overview 6
- USB Interface 6
- Power Configuration 7
- I/O Interface 8
  - Initial Values 9
  - Pin Groups 9
- Microcontroller 10
- Firmware Reset 10
- Main Application 10
- Bootloader 10

4 Software Support **12**
- The Big Picture 12
- AxiCat Server 13
- AxiCat Client for Python 14
- AxiCat Command Line Tool 14
- AxiCat Application Layer 15
- MCU Bootloader Tool 15
- AbioCard GUI 15
- AbioCard Server 15
- 1-Wire Automation Server 16

5 Application Examples **17**
- Connecting Things 17
  - GPIO 17
  - UART 18
  - TWI Master 20
  - TWI Slave 21
  - SPI Master 24
  - 1-Wire Master 25
- Deploying as Development Board 27
- Connecting an AbioCard to any Supported Computer 28
- Connecting an AbioWire to any Supported Computer 29
- I2C Multi-Master Arbitration Test Setup 30

6 Serial Paths **32**
- Linux 32

Windows 33

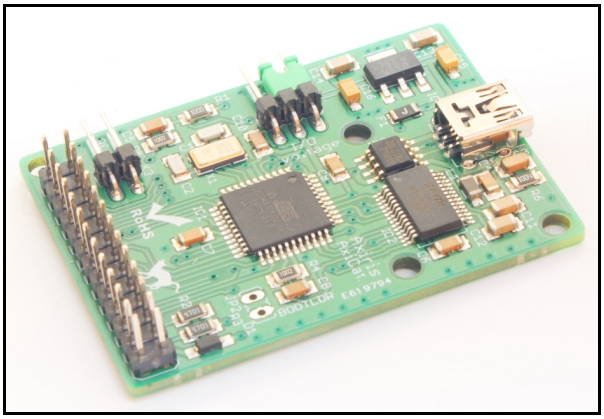
7 Legal Information 34
Disclaimer 34
Trademarks 34

8 Contact Information 34

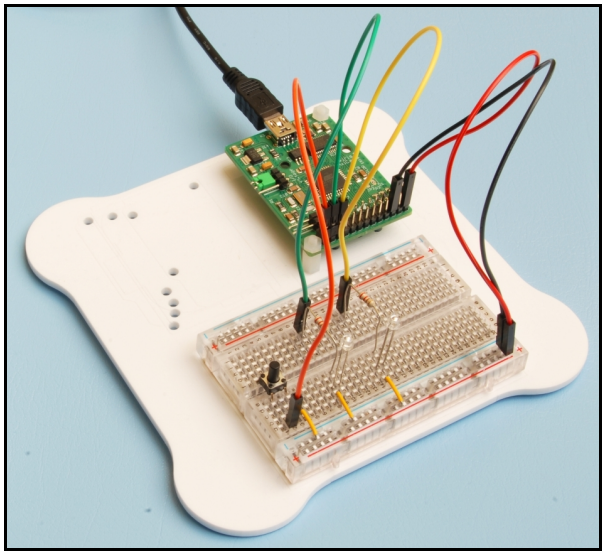
Revision History

Date	Authors	Description
2014-07-11	Peter S'heeren	Initial release.
2014-09-05	Peter S'heeren	Updated software section. Updated application examples for use with AxiCat server. Second release.
2014-09-17	Peter S'heeren	Added 1-Wire. Third release.
2014-09-28	Peter S'heeren	Fourth release.
2014-12-12	Peter S'heeren	Fifth release.

1 Features



- Multipurpose I/O card with USB interface.
- Powerful 12 MHz ATmega164A/324A/644A/1284 microcontroller.
- Selectable 3.3 V and 5 V I/O voltage levels.
- 17 bidirectional general-purpose I/O pins with programmable pull-up.
- 2 fully independent serial ports.
- SPI bus, master function, 4 slave select lines.
- I2C bus with pull-up resistors, master and slave function, support for repeated start, arbitration, clock stretching, device probing.
- 1-Wire bus, master function, strong pull-up control, accelerated enumeration and probing of 1-Wire slaves.
- Truly asynchronous design enables all interfaces to operate concurrently while executing I/O commands back-to-back avoiding inter-command delays.
- USB bus-powered.



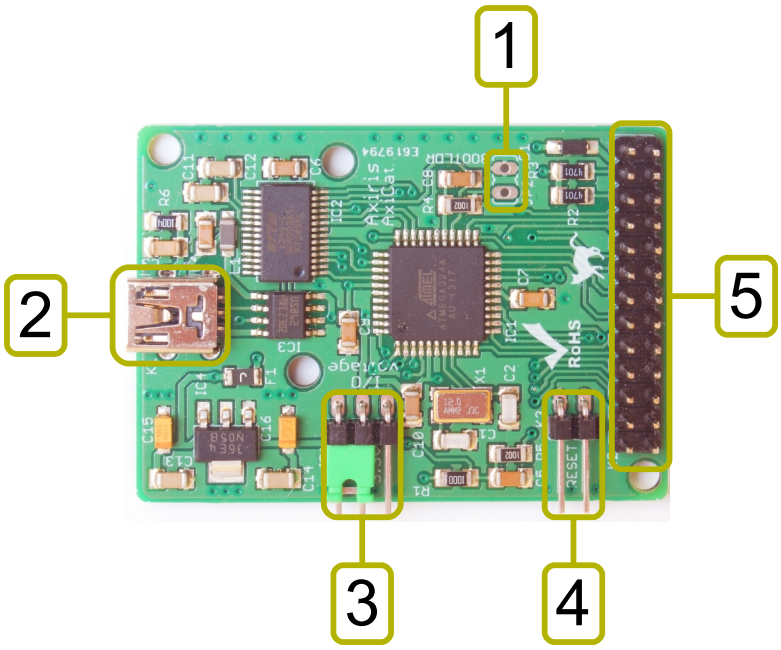
- Support for USB remote wake-up.
- Can be used as general-purpose development board for Atmega164A/324A/644A/1284 microcontrollers.
- Integrated bootloader and accompanying software for Linux and Windows enable easy upgrading of firmware and development of own software.
- Four mounting holes.
- Prototyping of I2C slave functions.
- Educational platform for learning and teaching I2C.
- Allows you to connect AbioCard, AbioWire and other boards to a PC.
- The AxiCat server program enables network and local access using an easy protocol.
- Use Python, PHP, C, C++, or other languages to program the card.

2 Technical Specifications

Weight	12 g
Dimensions	56 mm x 40 mm x 12 mm (W x D x H)

3 AxiCat

Board Overview



Mark	Label	Description
1	BOOTLDR, JP2	Close pads to activate bootloader on reset or power-up.
2	K1	USB Mini-B receptacle.
3	JP1	Jumper for selecting between 3.3 V and 5 V on K2.
4	RESET, K3	Close pins to force a firmware reset.
5	K2	Header that exposes the various I/O interfaces.

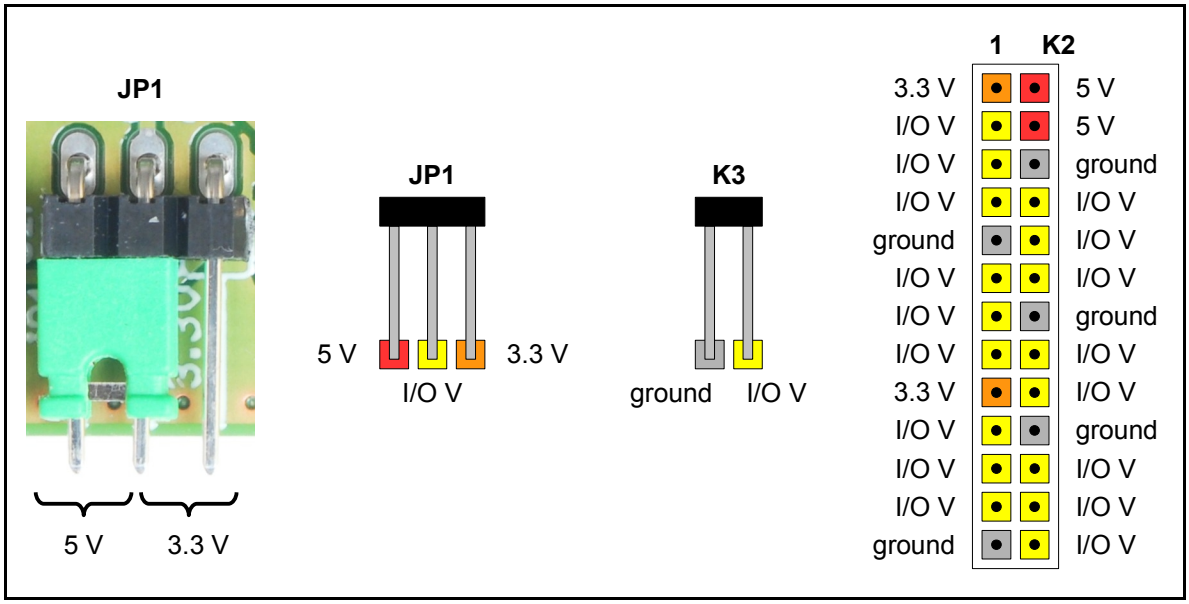
USB Interface

The AxiCat incorporates an FTDI FT245 USB-to-FIFO chip. Once enumerated, the USB function will appear as a serial port in your operating system. The operating system assigns a serial path to the serial port.

The board is USB bus-powered. The maximum current provided by a USB port is 500 mA. The AxiCat has a polyfuse that limits the current usage to 350 mA.

The AxiCat supports USB remote wake-up. This means the device can wake the USB function, and possibly the entire system, when certain events occur on the I/O interface, like the arrival of data on one of the UARTs.

Power Configuration



The I/O interface (K2) has the capability to work at two different voltage levels: 3.3 V and 5 V. The setting of the jumper (JP1) determines the voltage level.

The picture above shows the jumper setting for 5 V operation.

If you remove the jumper, the USB function will enumerate when you plug in the AxiCat, but the I/O interface will be non-operational.

IMPORTANT!

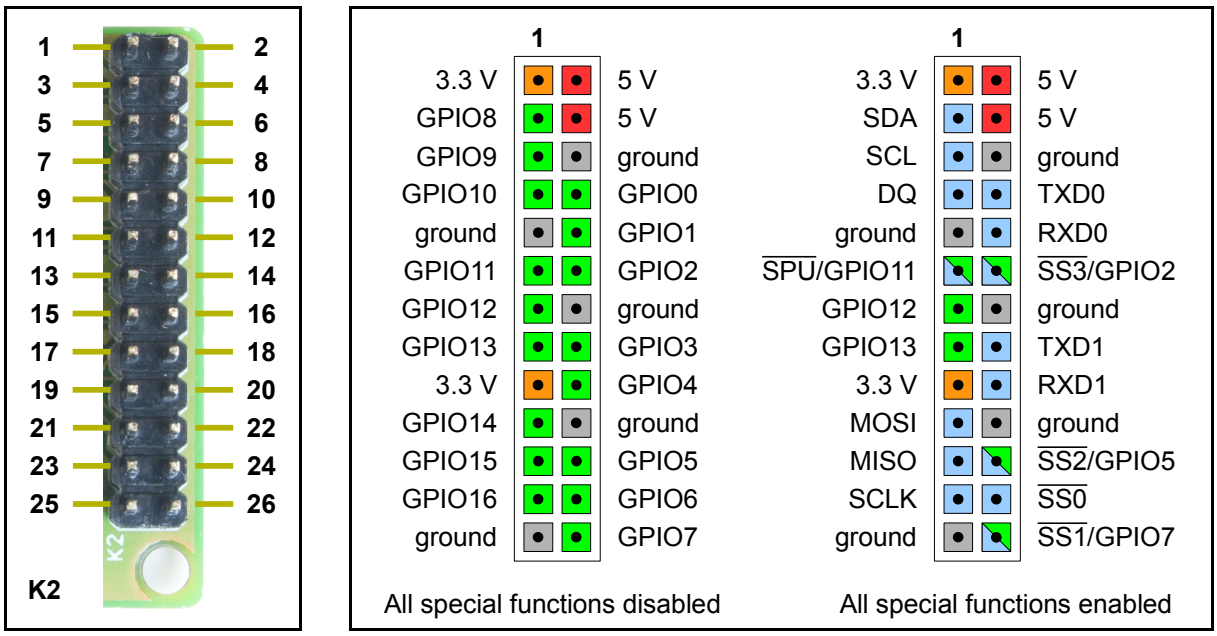
- Always check the position of the jumper (JP1) before you power up the AxiCat. Be sure the selected voltage matches the voltage level of the hardware that's attached to the I/O interface.
- The jumper must be set into position only when the AxiCat is powered off. If you change the jumper while the AxiCat is powered on, the result will be unpredictable and you may destroy hardware that's connected to the board.

When attaching hardware to the AxiCat, it's important to understand how the power lanes are routed between the board's various pins. The picture above uses colored squares to denote the wiring of the pins:

- Ground (gray): All ground pins are wired together including the ground line of the USB cable.
- 5 V (red): The 5 V comes from the USB cable and is distributed to all 5 V pins.
- 3.3 V (orange): This voltage is derived from the 5 V supply and distributed to all 3.3 V pins.
- I/O V (yellow): The I/O voltage lane is wired to either the 3.3 V or 5 V by means of the jumper JP1. All I/O pins (K2) and the reset input pin (K3) operate at this voltage.

I/O Interface

The I/O interface (K2) consists of 26 pins. The pin numbering is shown in the left picture. Pin number one is marked as a reference in various illustrations.



Seventeen pins are digital general-purpose input/output (GPIO) pins. Each GPIO pin can be individually programmed as input or output. A pull-up resistor can be enabled and disabled for each input pin. All GPIO pins operate at the voltage level selected on JP1.

Note that GPIO8 and GPIO9 are hardwired to 4.7 KΩ pull-up resistors on the AxiCat board. These pins are part of the I2C bus which requires pull-up resistors.

Besides the GPIO pins, the I/O interface offers four special functions: UART0, UART1, TWI and SPI. Each special function can be individually enabled and disabled. When a special function is enabled it overrides a group of GPIO pins. Most GPIO pins share a special function. All special functions operate at the voltage level selected on JP1.

UART0 and UART1 provide a serial bus with full duplex operation. The UARTs operate completely independent. Supported features include 5-9 data bits, 1-2 stop bits, standard baud rates up to 115200 baud, and raw speed setting.

The Two-Wire Interface (TWI) implements a full-fledged I2C bus. Master and slave operations are possible. Features include repeated start, clock stretching, multi-master arbitration, standard speeds of 100 kHz and 400 kHz, and raw speed setting.

The Serial Peripheral Interface (SPI) implements a three-wire synchronous serial bus with master operation and four slave select lines.

Note that three of the SPI slave select lines ($\overline{SS1}$, $\overline{SS2}$ and $\overline{SS3}$) do not override the functionality of the GPIO pins. You can use any of these lines as GPIO as long as you don't specify them in an SPI command. If you plan to use one or more of these slave select line for SPI, you have to initialize the corresponding GPIO pins to output and logic one before sending SPI commands.

The 1-Wire interface provides a 1-Wire bus with master operation and optional control for an external strong pull-up.

Initial Values

GPIO[0..16]	
Direction	Input
State	Pull-up active, logic 1

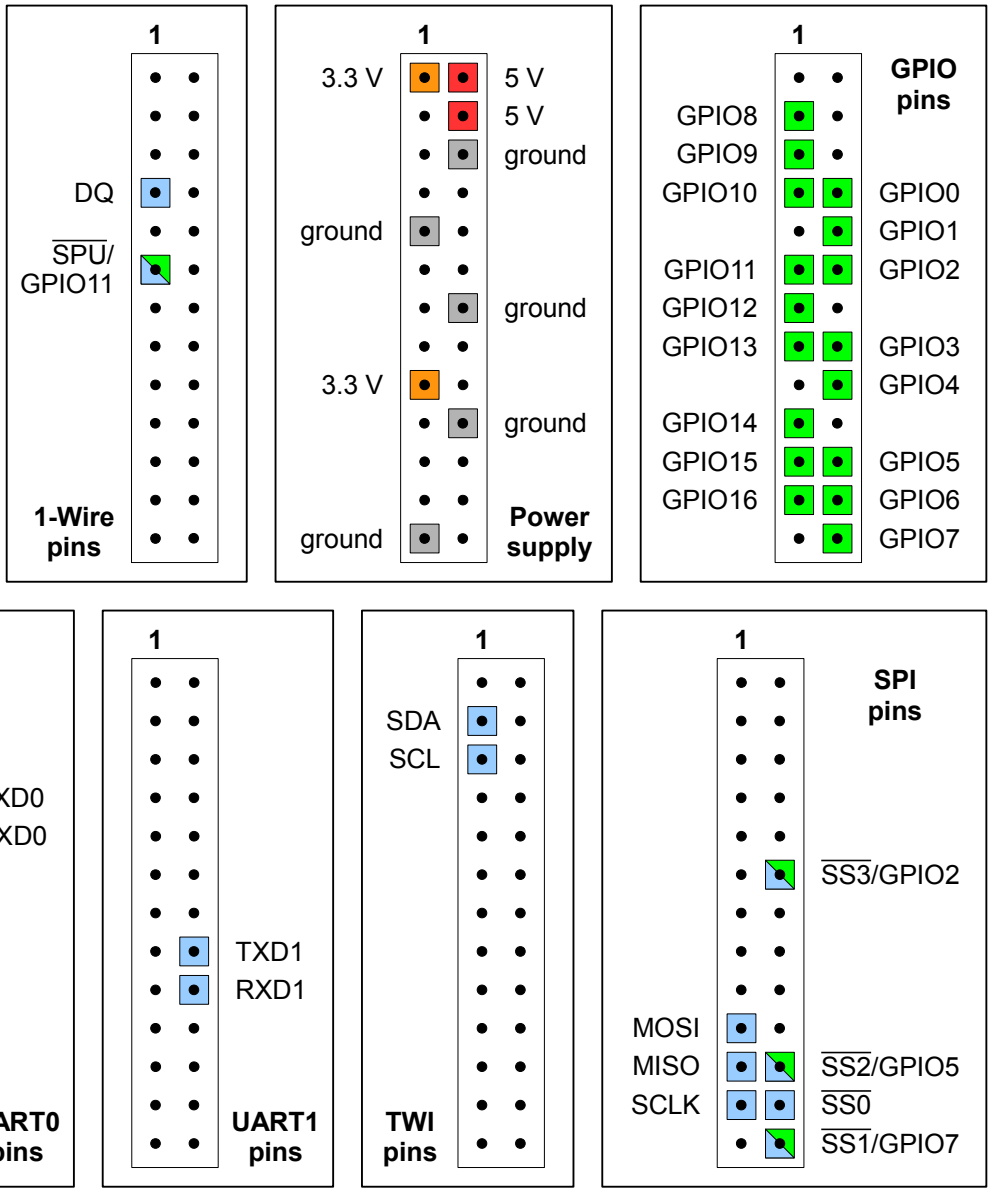
UART[0..1]	
Function	Disabled
Baud rate	115200
Data bits	8
Stop bits	1
Rx timeout	50 ms

TWI	
Function	Disabled
Speed	100 kHz
Slave	Disabled
Address	0000000b
General call	Disabled

SPI	
Function	Disabled
Speed	750000 Hz

1-Wire	
Function	Disabled
Speed	Standard

Pin Groups



Microcontroller

The AxiCat's design incorporates a microcontroller from the ATMEL® Atmega164A/324A/644A/1284 family. These chips are pin-compatible and offer different sizes in Flash memory, EPPROM and SRAM. The microcontroller is clocked at 12 MHz.

A standard AxiCat is issued with an ATmega164A microcontroller. This processor is more than capable of providing all functionality offered in the standard version.

It's possible to develop and deploy your own software for the AxiCat. When the application exceeds the memory limits of the ATmega164A, AxiCat can be equipped with a more powerful microcontroller.

Note: Axiris may deliver an AxiCat with a more powerful microcontroller at its own discretion.

Firmware Reset

You can reset the firmware in a powered AxiCat by shorting the pins of the RESET/K3 header. The firmware will restart as soon as the pins are released.

There are several instances where you may want to reset the firmware:

- Activate the bootloader.
- Quickly revert the I/O interface and main application to their initial state.

Main Application

The main application implements the core functionality of the AxiCat. The AxiCat is shipped with the main application preprogrammed in the flash memory of the microcontroller.

The main application accepts incoming command packets and issues response packets over the USB interface. Command packets are translated to activities on the I/O interface or other actions like setting parameters. Response packets are sent when the I/O interface has completed activities or status must be reported.

Command and response packets and their behavior are part of the AxiCat serial protocol. The serial protocol is specified in a separate document.

For more information, please read AxiCat Technical Note 001 - Serial Protocol. This document is available for download on the Axiris website.

Bootloader

The AxiCat comes with a bootloader preprogrammed in the flash memory of the microcontroller. The bootloader offers several features:

- You can upgrade to the latest version of the main application.
- As a maker, you can develop your own main application and use the bootloader to download and run your software in the AxiCat.
- It provides information about your AxiCat.
- It works together with the free **aximcubldr** software.

The bootloader is the first piece of software that's executed when the AxiCat is powered up. As soon as the bootloader is started, it checks the connection of the pads labeled JP2/BOOTLDR.

When the pads are open, the bootloader hands over control to the main application which offers the main features of the AxiCat. This is the normal, intended operation of the board.

When the pads are closed, the bootloader remains active. It takes over the USB interface and as such the associated serial port in your operating system communicates with the bootloader instead of the main application.

To activate the bootloader, follow these steps:

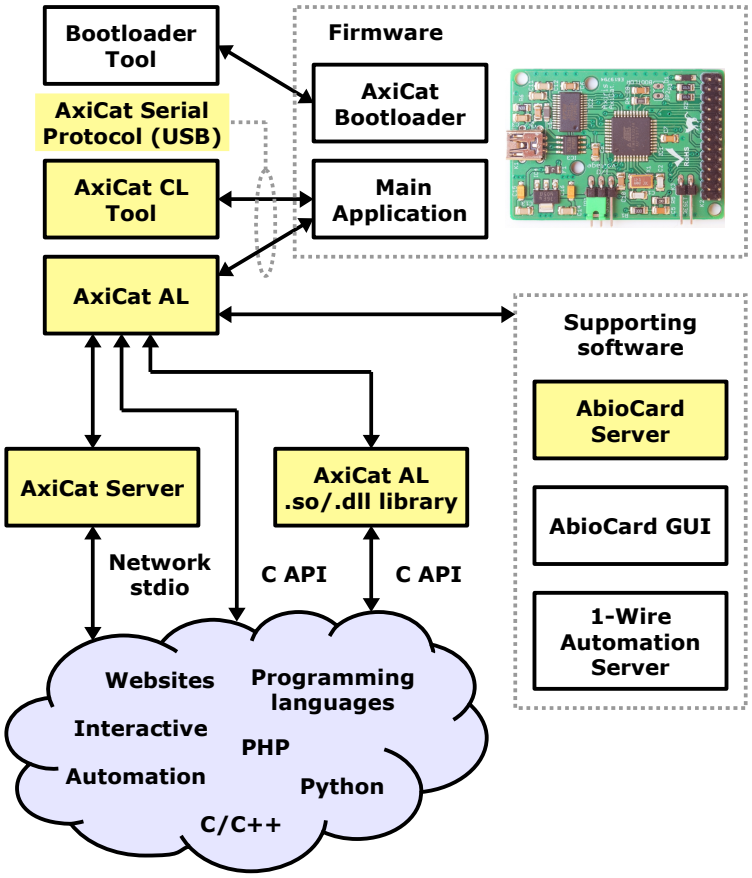
1. Close the pads labeled JP2/BOOTLDR. You can use a male-male jumper wire or similar.
2. If the board is powered up, reset the firmware, else connect the board to the USB bus to power it up.

Once the bootloader is activated, you can use the **aximcubldr** program to communicate with the bootloader from the computer the AxiCat is plugged into. The program can examine the bootloader, write HEX object files into flash memory and EEPROM of the microcontroller, and run the firmware application right away.

4 Software Support

The Big Picture

Software support for AxiCat is quite elaborate. The versatile nature of the AxiCat has lead to the creation of various software components. The following picture gives an overview of the relationship between the software components:



All software runs on Linux and Windows operating systems. This way the end-user is given a great deal of flexibility and choice of computer systems and configurations when deploying the AxiCat.

Source code of the parts marked in yellow is available.

AxiCat Server

This program is an important tool for working with the AxiCat. It's usually run as a server process hence the name. It offers the functionality of the AxiCat Application Layer as a server.

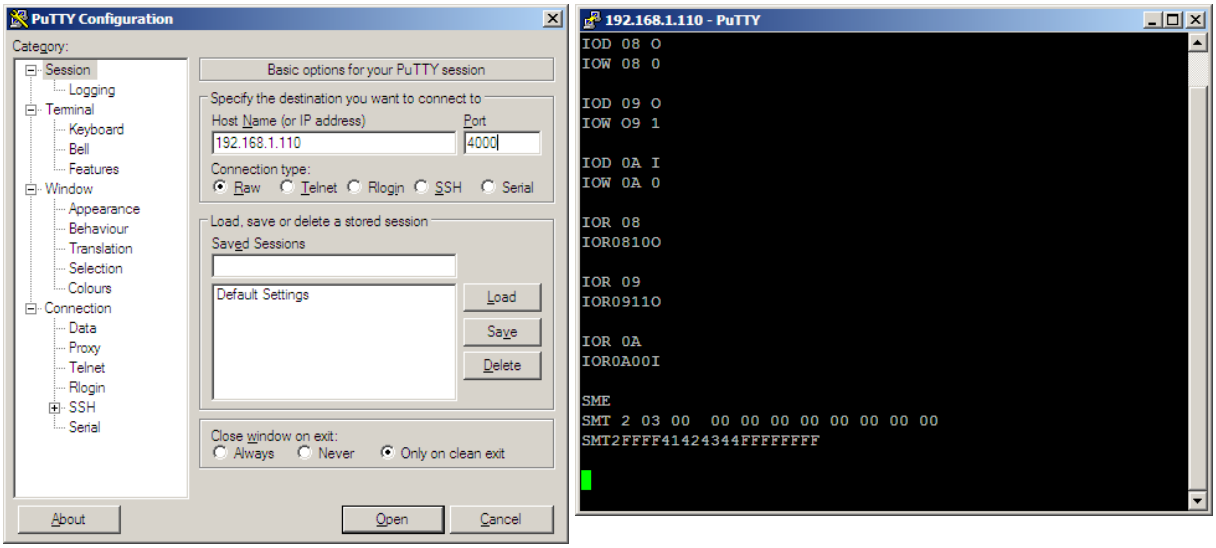
The server offers two modes of communication with the AxiCat:

- Network mode: a socket interface enables communication over the network.
- Standard input and output mode: another process communicates with the AxiCat by means of redirecting its input and output to the AxiCat server.

The server uses a simple communications protocol that is both human-readable and easy to format and parse programmatically. Commands and responses are composed of ASCII characters.

Commands can be padded with spaces to enhance readability. Empty lines and comments are permitted as well. The latter features come in handy when you create script files.

You can manually send commands using a terminal program, like PuTTY. For example:



In the above example, the server is connected to from a remote computer. All it takes is the IP address and port of the server. Next, a number of commands are typed in to which the server responds.

For the curious, the SPI transfer actually reads out the first 8 bytes from a Microchip 25LC020A EEPROM chip. The $\overline{CS}$ pin of the EEPROM is connected to $\overline{SS2}$ on the AxiCat. The first four bytes had been programmed as "ABCD" earlier.

Network mode is great for controlling the AxiCat from a programming language. Every popular language has built-in support for programming with network sockets. The format of commands and responses allows for very simple processing in your language of choice.

The following example shows how to use the AxiCat from a Python program:

```
import axicatclient

# Connect to the AxiCat Server
client = axicatclient.AxicatClient("localhost",4000)

# Set GPIO 1 as input, high state meaning pull-up activated
client.gpioSetDir(1,client.GPIO_DIR_INPUT)
client.gpioWrite(1,1)

# Read input state of GPIO 1
state = client.gpioRead(1)
print "GPIO #1: input state %i" % state
```

Save this file as **client.py** for example. You can run the program in Linux as follows:

```
$ python client.py
GPIO #1: input state 1

$
```

The output shows that the Python program successfully connected to the server, sent the GPIO read command, and received the response containing the state of GPIO pin 1.

For more information, please read the AxiCat Server user manual. The manual is available for download on the Axiris website.

AxiCat Client for Python

This module is written in Python. It enables Python programs to communicate with the AxiCat Server as a network client. The previous section contains an example Python program that uses the module.

The module offers a blocking API. Since the AxiCat Server defines an asynchronous communications protocol, the Python module can't offer the full functionality of the server. On the other hand, the blocking API of the Python module is very easy to use and looks familiar to programmers who are used to working with similar third-party Python modules.

AxiCat Command Line Tool

This program talks directly with the AxiCat at the level of the serial interface. This is the lowest level of communication possible and is defined as the AxiCat serial protocol.

The program applies the serial protocol in a very simple and minimalistic way. It takes user commands and translates them into command packets that are sent to the AxiCat. A number of user commands read back data bytes from the AxiCat.

Other user commands provide features such as reading data bytes from the AxiCat and waiting for a certain amount of milliseconds.

For more information, please read the AxiCat Command Line Tool user manual. The manual is available for download on the Axiris website.

AxiCat Application Layer

The AxiCat Application Layer (AL) logically sits on top of the AxiCat serial protocol. It's a software module that implements the complete serial protocol and offers a non-blocking API to the upper layer.

Thanks to its non-blocking design, the module offers high performance for all available I/O. You can easily schedule multiple transfers for GPIO, I2C, SPI, 1-Wire and UARTs concurrently; the module will efficiently handle all transfers with the AxiCat and report completion when ready.

The AxiCat AL is available as C source code so you can incorporate the module into your own software that uses the AxiCat. The source of program **axicatserver** is a good starting point for learning to develop with the module.

MCU Bootloader Tool

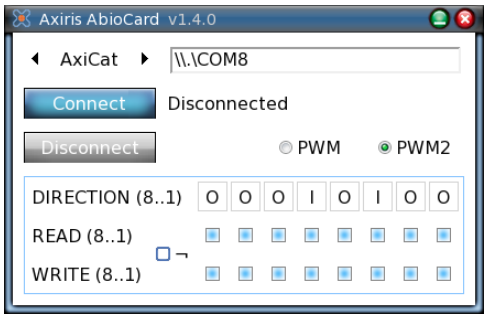
Program **aximcubldr** v1.0.1 or later adds support for the AxiCat bootloader. This tool allows you can upgrade the AxiCat with the latest version of the main application.

The tool also enables you to deploy the AxiCat as a general-purpose development board for the family of ATmega164A/324A/644A/1284 microcontrollers.

AbioCard GUI

The **abiocardgui** program starting from v1.4.0 adds support for the AxiCat.

With the addition of the AxiCat interface, it's now possible to interactively work with an AbioCard model A or B on any computer that's supported by the AbioCard GUI program.



AbioCard Server

Program **abiocardserver** has been extended with support for AxiCat. It now runs on Linux as well as Windows.

This means the possibilities to deploy the AbioCard with the server program have been significantly increased.

You can run the server to control an AxiCat with AbioCard combo on any supported Linux system, like this:

```
# ./abiocardserver -axicat /dev/ttyUSB0 -p 5678
```

Likewise, in Windows:

```
> abiocardserver.exe -axicat \\.\COM4 -p 5678
```

You can run multiple instances of the server on a single computer, each server instance controlling an AxiCat with AbioCard combo. For example:

```
# ./abiocardserver -axicat /dev/ttyUSB0 -p 5678 &
# ./abiocardserver -axicat /dev/ttyUSB1 -p 5679 &
```

Running the equivalent server instances in Windows may look like this:

```
> abiocardserver.exe -axicat \\.\COM4 -p 5678
> abiocardserver.exe -axicat \\.\COM5 -p 5679
```

1-Wire Automation Server

The 1-Wire Automation Server supports special functions 1-Wire, UART0, UART1 and TWI. You can connect 1-Wire slaves directly to the 1-Wire bus. You can hook up a variety of 1-Wire adapters with an I2C or serial interface.

It's possible to connect multiple 1-Wire adapters to a single AxiCat. For example, you could connect a HA7S to UART0, another HA7S to UART1, a set of DS2482 chips to the I2C bus, and a couple of DS18B20 temperature sensors to the 1-Wire bus. Just make sure that all connected components operate at the same voltage level.

5 Application Examples

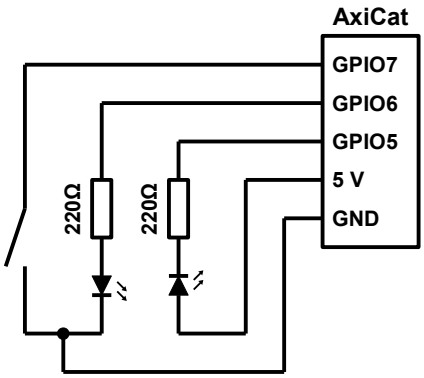
Connecting Things

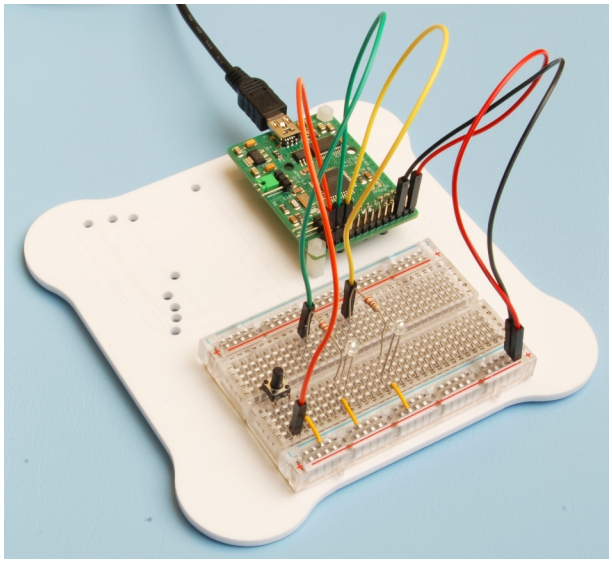
The AxiCat is very well suited for connected and exploring an endless variety of hardware ranging from simple I/O to a complex synergy of all special functions. This section contains a number of examples to start with. Each example highlights a specific subset of the functionality. You can combine the various special functions and features to create powerful solutions.

GPIO

Each GPIO pin can be set as output or input. Each output buffer has symmetrical drive characteristics with both high sink (output low) and source (output high) capability. Each input has a selectable pull-up resistor with a supply-voltage invariant resistance. An input without active pull-up resistor behaves as tri-state. All GPIO pins have protection diodes to ground and Vcc. Please refer to the Atmega164A/324A/644A/1284 datasheet for more information.

The following example demonstrates the capabilities of the GPIO pins. It's assumed that program **axicatserver** is connected to the AxiCat in the example setup.





The setup uses GPIO5 as a sinking output, GPIO6 as a sourcing output and GPIO7 as an input with active pull-up.

First you've to initialize the GPIO pins. Send the following commands to the server:

Command	`IOD 05 O`	Set GPIO5 as output
Command	`IOD 06 O`	Set GPIO6 as output
Command	`IOD 07 1`	Set GPIO7 as input
Command	`IOW 07 1`	Activate pull-up on GPIO7

Let's drive GPIO5. Set the output low (logic 0) to sink current and turn on the LED. Set high (logic 1) to turn off the LED.

Command	`IOW 05 0`	Turn on LED on GPIO5
Command	`IOW 05 1`	Turn off LED on GPIO5

GPIO6 is used as sourcing output. Set the output high (logic 1) to source current and turn on the LED. Set the output low (logic 0) to turn off the LED.

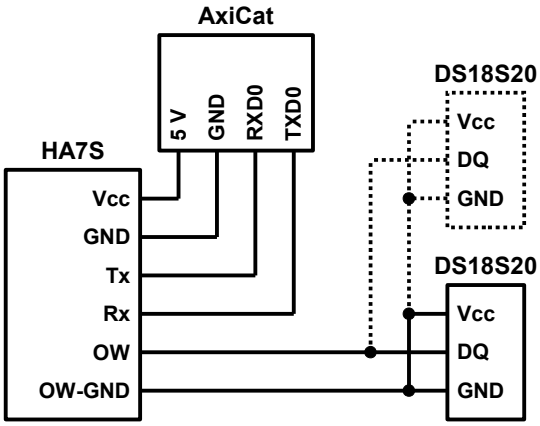
Command	`IOW 06 0`	Turn off LED on GPIO6
Command	`IOW 06 1`	Turn on LED on GPIO6

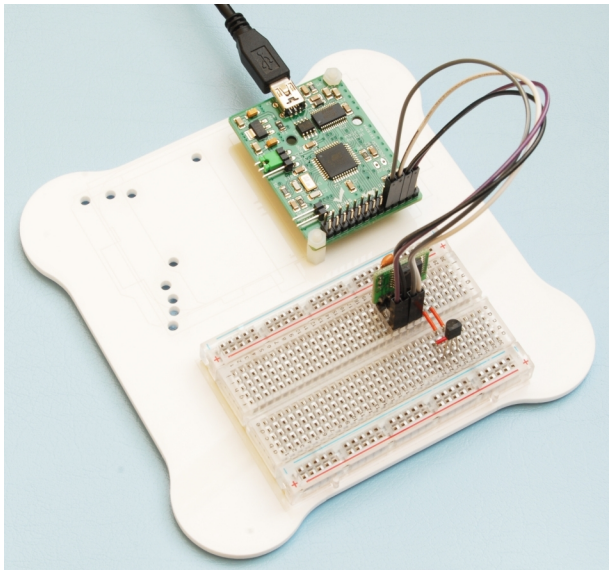
A switch controls the input state of GPIO7. When the switch is closed, the input is pulled to ground. When left open, the active pull-up pulled the input to Vcc.

Command	`IOR07`	Read state of GPIO7
Response	`IOR0711I`	Input sensed state is high, meaning the switch is open
Command	`IOR07`	Read state of GPIO7
Response	`IOR0701I`	Input sensed state is low, meaning the switch is closed

UART

This section explains how to connect things to the UARTs. We've chosen a practical example that features a simple 1-Wire network. It's assumed that program **axicatserver** is connected to the AxiCat in the example setup.





The image shows a HA7S 1-Wire adapter attached to UART0 of the AxiCat. The HA7S is the small PCB sticking out of the breadboard. The 1-Wire adapter uses a serial interface. The serial settings are 9600 baud, 8 data bits, 1 stop bit. The communications protocol employs a simple, synchronous command-response model using ASCII characters.

A DS18S20 1-Wire temperature sensor is wired to the HA7S. The temperature sensor comes in a TO-92 package that resembles a transistor. In this example the DS18S20 is wired for parasite power mode, which means it taps power from the DQ line rather than the Vcc line. Parasite power mode allows one to limit the 1-Wire bus to merely two wires

(DQ and ground).

The entire setup operates at 5 V. The AxiCat powers the 1-Wire adapter and temperature sensor.

The dotted area in the schematic indicates you can connect additional 1-Wire slaves to the 1-Wire adapter. Note that not all 1-Wire slaves offer parasite power mode and need an external 5 V power supply instead. You can use the 5V line of the AxiCat. If you connect many 1-Wire slaves, you'll hit the power limit of the AxiCat at some point. In that case you can resort to deploying a 1-Wire power injector.

Here's how to enumerate the 1-Wire slave device(s) connected to the HA7S.

Command	**U0S D 8**	Set 8 data bits
Command	**U0S S 1**	Set 1 stop bit
Command	**U0S B 9600**	Set 9600 baud
Command	**U0E**	Enable UART 0
Command	**U0W 53**	Transmit "SEARCH ROM" command to HA7S, Tx data: "S"
Command	**U0R 0020**	Read data from UART
Response	**U0R42313030303830324134 3941313731300D**	UART0 received 17 (10h+1) data bytes Rx data: "B1000802A49A1710" CR HA7S reports ROM code 10-000802A49A17-B1
Command	**U0W 73**	Transmit "NEXT SEARCH ROM" command to HA7S, Tx data: "s"
Command	**U0R 0020**	Read data from UART
Response	**U0R0D**	UART0 received 1 (00h+1) data byte Rx data: CR (empty string)

The table shows an enumeration of one 1-Wire slave. If your setup includes additional 1-Wire slaves then you can repeat the "s" command to retrieve the ROM codes of all 1-Wire slaves.

Here's another enumeration with two 1-Wire slave devices connected to the HA7S. This time unsolicited responses are enabled meaning the server automatically sends UART Rx data to the client.

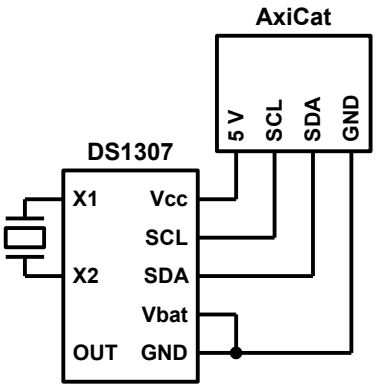
Command	**U0S D 8**	Set 8 data bits
Command	**U0S S 1**	Set 1 stop bit
Command	**U0S B 9600**	Set 9600 baud
Command	**U0E**	Enable UART 0
Command	**U0S U 0020**	Enable unsolicited responses, 32 data bytes
Command	**U0W 53**	Transmit "SEARCH ROM" command to HA7S, Tx data: "S"
Response	**U0R42313030303830324134 3941313731300D**	UART0 received 17 (10h+1) data bytes Rx data: "B1000802A49A1710" CR

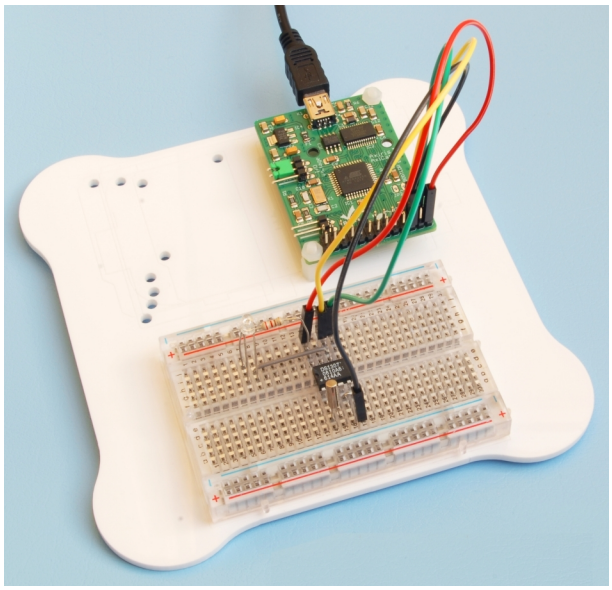
		HA7S reports ROM code 10-000802A49A17-B1
Command	`U0W 73`	Transmit "NEXT SEARCH ROM" command to HA7S, Tx data: "s"
Response	`U0R433430303030343043 4242423232380D`	UART0 received 17 (10h+1) data bytes Rx data: "C40000040CBBB228" CR HA7S reports ROM code 28-0000040CBBB2-C4
Command	`U0W 73`	Transmit "NEXT SEARCH ROM" command to HA7S, Tx data: "s"
Response	`U0R0D`	UART0 received 1 (00h+1) data byte Rx data: CR (empty string)

TWI Master

The AxiCat offers an elaborate I2C master function. The example presented in this section introduces the reader to the basic operations of the master.

We've attached a DS1307 real-time clock to an AxiCat. It's assumed that program **axicatserver** is connected to the AxiCat in the example setup.





The slave address of the DS1307 is 1101000b. Let's check whether the DS1307 reacts to the slave address:

Command	`IMT D0`	SLA+W, address 68h, no data bytes
Response	`IMTD00000A`	The slave acknowledged SLA+W

The response affirms the DS1307 is present on the I2C bus. Note that if the DS1307 weren't present, the response would be **IMTD00000N**.

None of the registers are in a defined state after the DS1307 has powered up. Initialization is required. The following I2C SLA+W transfer programs a time and date, sets 24-hour mode, and instructs the real-time clock to start running (CH=0):

Command	`IMT D0 00 00 00 00 01 01 01 00 00`	Transmit 9 data bytes: register 00h, followed by eight bytes: CH=0, 00 seconds, 00 minutes, 00 hours, 24-hour mode, day 1, date 01, month 01, year 00, RS[1..0]=00b, SQWE=0, OUT=0
Response	`IMTD00009A`	All data bytes were transmitted

In order to read out the time and date, you've to write (SLA+W) the index of the first register to access, and read (SLA+R) the time and date registers:

Command	`IMT D0 00`	Write index of register to access
Response	`IMTD00001A`	One byte written to the slave
Command	`IMT D1 0007`	Read seven bytes
Response	`IMTD1A56160001010100`	The master received seven data bytes

The last response contains the time and date. These values will be different each time you read them out.

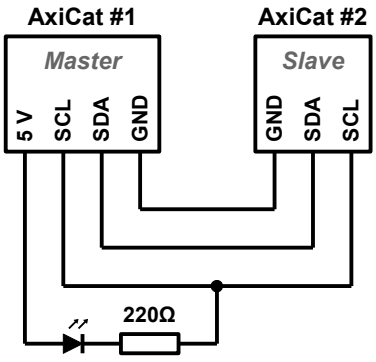
You can send the two I2C transfers at once. Add 'R' to the first command; doing so also ensures the second I2C transfer starts with a REPEATED START signal.

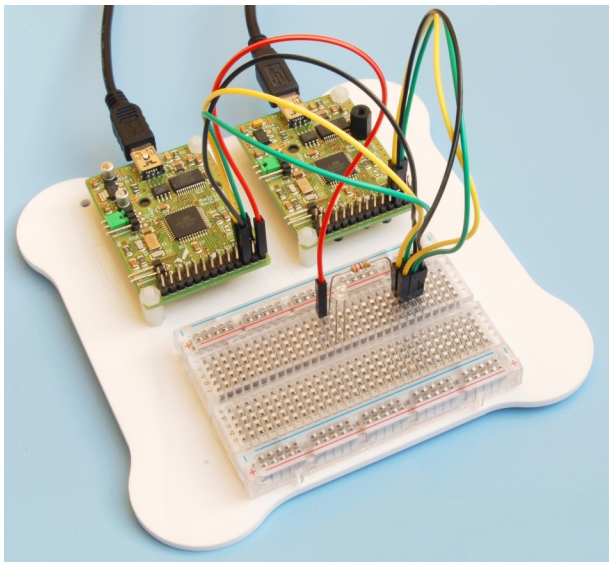
Command	`IMT D0 00 R`	Write index of register to access
Command	`IMT D1 0007`	Read seven bytes
Response	`IMTD00001A`	One byte written to the slave
Response	`IMTD1A56160001010100`	The master received seven data bytes

TWI Slave

The slave functionality is quite elaborate. The following example explores the basics of TWI slave function.

We've chosen a second AxiCat to act as the I2C master. You can use another I2C master instead, as long as it is able to detect clock stretching and act accordingly.





An I2C slave has the ability to pull SCL low. The slave literally "freezes" the I2C bus as no activity will occur as long as SCL is pulled low. This mechanism allows the I2C slave to buy some time when it can't respond to a master immediately.

For this example, we've wired a LED to the clock line. The LED will light up when SCL is low. During normal I2C bus activity, you may see the LED flicker briefly. When the slave pulls down SCL for a long period, the LED lights up clearly.

Normally one doesn't connect a LED to the SCL line. The capacitance of the LED may bring down the maximum bus speed so we use a speed setting of 50 kHz to be on the safe side.

It's assumed that an instance of program **axicatserver** is connected to each AxiCat in the example setup. Each instance has to use a different port number. Here's how to set up two servers in network mode for use with a terminal program (parameter **-crlf** is usually specified for use with terminal programs). Example in Linux:

```
$ ./axicatserver -s /dev/ttyUSB0 -p 4000 -crlf &
$ ./axicatserver -s /dev/ttyUSB1 -p 4001 -crlf &
```

Likewise, in Windows:

```
$ axicatserver -s \\.\COM10 -p 4000 -crlf &
$ axicatserver -s \\.\COM11 -p 4001 -crlf &
```

First things first. Let's set up the I2C function in master and slave. The I2C slave is assigned address 20h.

AxiCat #1 (Master):

Command	**IMS S 50000**	Set bus speed to 50000 Hz
Command	**IME**	Enable I2C master

AxiCat #2 (Slave):

Command	**IME**	Enable I2C master
Command	**ISE40**	Enable slave, address 20h, no GCA (general-call address)

Let's starts off with a simple example. The slave is programmed to return four characters "ABCD" when the I2C master reads from (SLA+R) the slave.

AxiCat #2 (Slave):

Command	**ISW 41 42 43 44 L**	Data bytes to write, full payload

The 'L' is important. If says the entire payload is specified.

AxiCat #1 (Master):

Command	**IMT 41 0004**	Read 4 bytes from address 20h
Response	**IMT41A41424344**	Response

The master reads four bytes from the slave. The response tells us that the slave acknowledged address 20h and four characters "ABCD" were transmitted. At this point, the I2C slave Tx transfer completes:

AxiCat #2 (Slave):

Response	**ISW0004N**	Four data bytes transmitted

This response indicates four bytes were transferred and the master signaled NACK after the last byte was transmitted. This is correct.

What if the master reads a different amount of data bytes? In case the master reads less bytes, the slave will get a response like **ISW0002A** (only 2 bytes were transmitted). If the master reads more bytes, the slave will pad the payload with FFh bytes, the response will be **ISW0004A**. Anyway, the response to the slave Tx transfer will contain 'A' which indicates the master read a different number of bytes than the slave provided.

Now let's pull SCL low. Proceed as follows:

AxiCat #1 (Master):

Command	**IMT 41 0004**	Read 4 bytes from address 20h

The LED lights up. The reason is that the I2C slave has been addressed successfully (SLA+R) but there's no data available to transmit, hence the slave pulls SCL low. The I2C bus is now totally locked up.

AxiCat #2 (Slave):

Command	**ISW 41 42 43 44 L**	Data bytes to write, full payload
Response	**ISW0004N**	Four data bytes transmitted

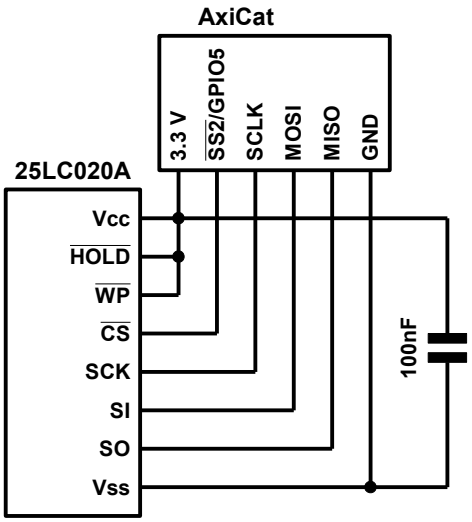
AxiCat #1 (Master):

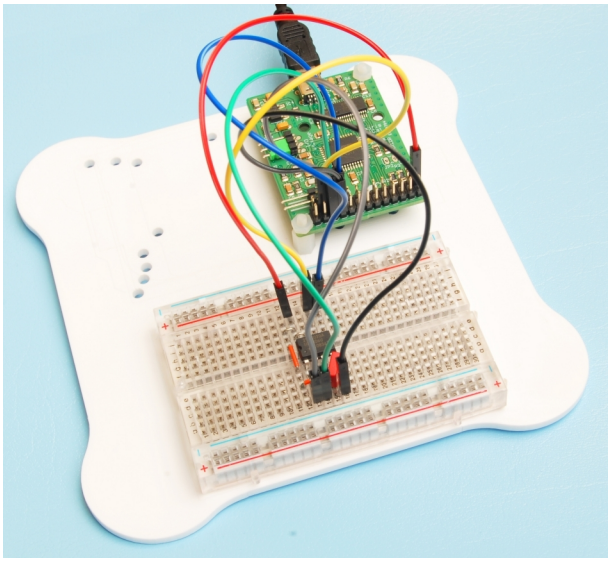
Response	**IMT41A41424344**	Response

As soon as we send the I2C slave Tx command, the I2C bus transfer between master and slave continues, the LED darkens and responses come back. The I2C bus becomes idle again.

SPI Master

The following example shows how to use the SPI master function with an EEPROM chip. We've picked the Microchip 25LC020A as the SPI slave. This EEPROM chip has a memory space of 256 bytes.





The circuit operates at 3.3 V. You can also make it work at 5 V, the SPI EEPROM chip has a Vcc range of 2.5 – 5.5 V.

It's assumed that program **axicatserver** is connected to the AxiCat in the example setup.

Initialization for this circuit requires setting up $\overline{SS2}$/GPIO5 and enabling the SPI master:

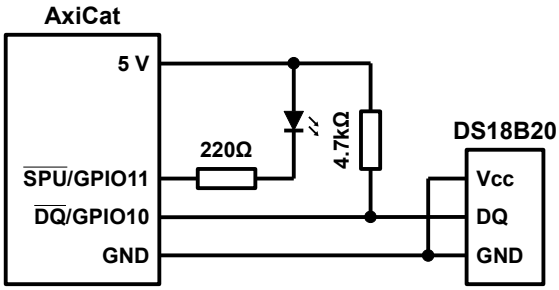
Command	`IOD 05 O`	Set $\overline{SS2}$/GPIO5 as output
Command	`IOW 05 1`	Set $\overline{SS2}$/GPIO5 high
Command	`SME`	Enable the SPI master

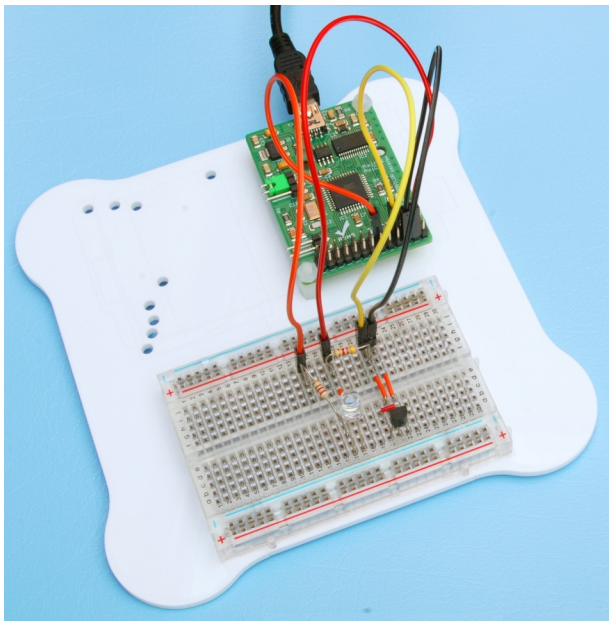
You can now start working with the EEPROM chip. Here are some example SPI transfers containing EEPROM commands:

Command	`SMT 2 05 00`	RDSR – read status register
Response	`SMT2FF00`	Status register is 00000000b
Command	`SMT 2 06`	WREN – enable write operation
Response	`SMT2FF`	
Command	`SMT 2 05 00`	RDSR – read status register
Response	`SMT2FF02`	Status register is 00000010b
Command	`SMT 2 02 20 45 46 47 48`	WRITE – write 4 bytes at offset 20h
Response	`SMT2FFFFFFFFFFFF`	
Command	`SMT 2 03 20 FFFFFFFFFFFFFFFF`	READ – read 8 bytes at offset 20h
Response	`SMT2FFFF45464748FFFFFFFF`	

1-Wire Master

The following example demonstrates the 1-Wire master function in combination with a DS18B20 temperature sensor.





The temperature sensor comes in a TO-92 package that resembles a transistor. The DS18B20 is wired for parasite power mode, which means it taps power from the DQ line rather than the Vcc line.

We're going to demonstrate the strong pull-up control mechanism as well. Rather than implementing a real pull-up circuit, we've hooked up a LED that lights up when $\overline{\text{SPU}}$ is active.

The circuit operates at 5 V. It's assumed that program **axicatserver** is connected to the AxiCat in the example setup.

First we set up $\overline{\text{SPU}}$/GPIO11 and enable the 1-Wire master:

Command	`IOD 0B O`	Set $\overline{\text{SPU}}$/GPIO11 as output
Command	`IOW 0B 1`	Set $\overline{\text{SPU}}$/GPIO11 high
Command	`OME`	Enable the 1-Wire master

Since there's only one 1-Wire slave present on the bus, we can use the Skip ROM command. Let's check the power mode of the DS18B20:

Command	`OMR`	1-Wire reset
Response	`OMR1`	Presence of 1-Wire slave detected
Command	`OMT CC B4`	Skip ROM → Read Power Supply
Response	`OMTCCB4`	
Command	`OMB 1`	Read a bit
Response	`OMB0`	DS18B20 reports a zero bit: parasite power mode

The DS18B20 reports parasite power mode. That's correct since we've wired the

DS18B20's power line to ground.

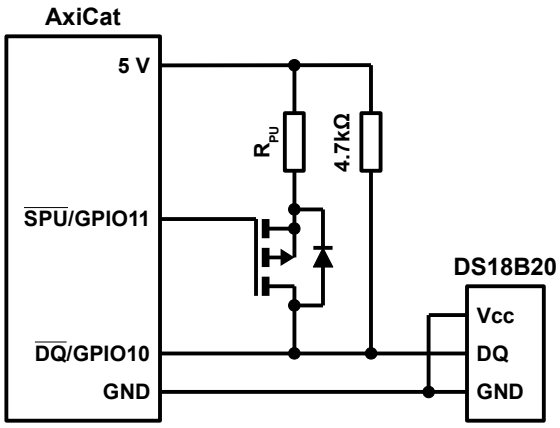
Start a temperature conversion and enable strong pull-up:

Command	**OMR**	1-Wire reset
Response	**OMR1**	Presence of 1-Wire slave detected
Command	**OMT CC 44 S**	Skip ROM → Convert Temperature, enable strong pull-up
Response	**OMTCC44**	

The LED will light up. The DS18B20 needs about 750 ms to convert the temperature. After that we can read the results from the tempearature sensor:

Command	**OMR**	1-Wire reset
Response	**OMR1**	Presence of 1-Wire slave detected
Command	**OMT CC BE FFFFFFFFFFFFFFFF**	Skip ROM → Read Scratchpad
Response	**OMTCCBEFF074B467FFF01102F**	Scratchpad data (8 bytes and CRC)

The DS18B20 returns 9 bytes, the last data byte being the CRC. The first two bytes are the 16-bit temperature register, LSB first. The register's value is 07FFh. This value is out of range and indicates that our particular DS18B20 couldn't convert the temperature properly. The reason is the lack of an actual strong pull-up circuit.



This is an example of a circuit with strong pull-up. The value of R_{PU} can be 180Ω.

The 1-Wire master incorporates a powerful enumeration command. Let's enumerate the DS18B20 sensor:

Command	**OMNF**	Search first slave
Response	**OMN28-0000040CBBB2-C4**	ROM code of the DS18B20
Command	**OMNN**	Search next slave
Response	**OMN**	No next slave found

The ROM code of the DS18B20 sensor is "28-0000040CBBB2-C4".

Deploying as Development Board

You can deploy your AxiCat as a general-purpose development board for the family of ATmega164A/324A/644A/1284 microcontrollers. You can write your own software that runs on the AxiCat. In doing so, you effectively replace the main application with your own creation.

You don't need a third-party programmer, the preprogrammed bootloader and the **aximcubldr** software are all you need.

The **avr-gcc** tool chain is good choice for developing software for the AxiCat. With this tool chain, which is available for Windows, Linux and other operating systems, you can write software in C/C++ source code. The output after building your source code is a HEX object file. With the help of the bootloader you can run your newly created software in the AxiCat.

After you've activated the bootloader, use the **aximcubldr** software to download and run your HEX object file in the AxiCat. In Windows:

```
> aximcubldr.exe -serial \\.\COM4 -flash myimage.hex -run
```

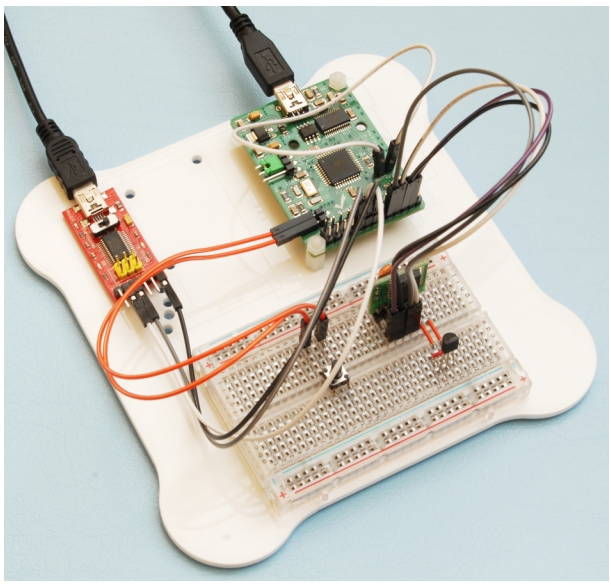
Likewise, in Linux:

```
$ ./aximcubldr -serial /dev/ttyUSB0 -flash myimage.hex -run
```

If you run the command without the -flash option, the bootloader jumps to the main application.

The image shows an example setup for developing a program that controls a HA7S 1-Wire adapter connected to UART0 on the AxiCat. The entire setup works at 5 V.

You probably want close the pads (JP2/BOOTLDR) for the duration of the development of your own software. You can use a male-male jumper wire for this purpose. It's the looped wire in the image.



We've wired a push button to the reset pins.

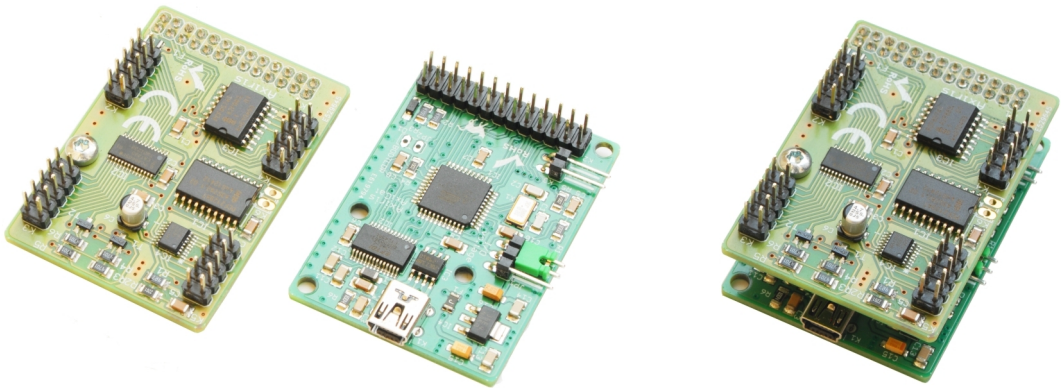
Since JP2 is closed, pressing the reset button will bring the AxiCat in bootloader mode. This behavior is convenient for development where you usually repeat this sequence a lot: reset → flash → run.

The AxiCat features two UARTs. You can hook up one UART to a 3.3 V or 5 V level serial port that is connected to a computer. With this setup, you can transmit debug information from your own software over the serial line to the computer and print the debug information in a terminal program like Putty. The image shows a USB-to-serial adapter with 3.3 V and 5 V selectable interface wired to UART1 on the AxiCat.

Note that you can always put back the original AxiCat main application.

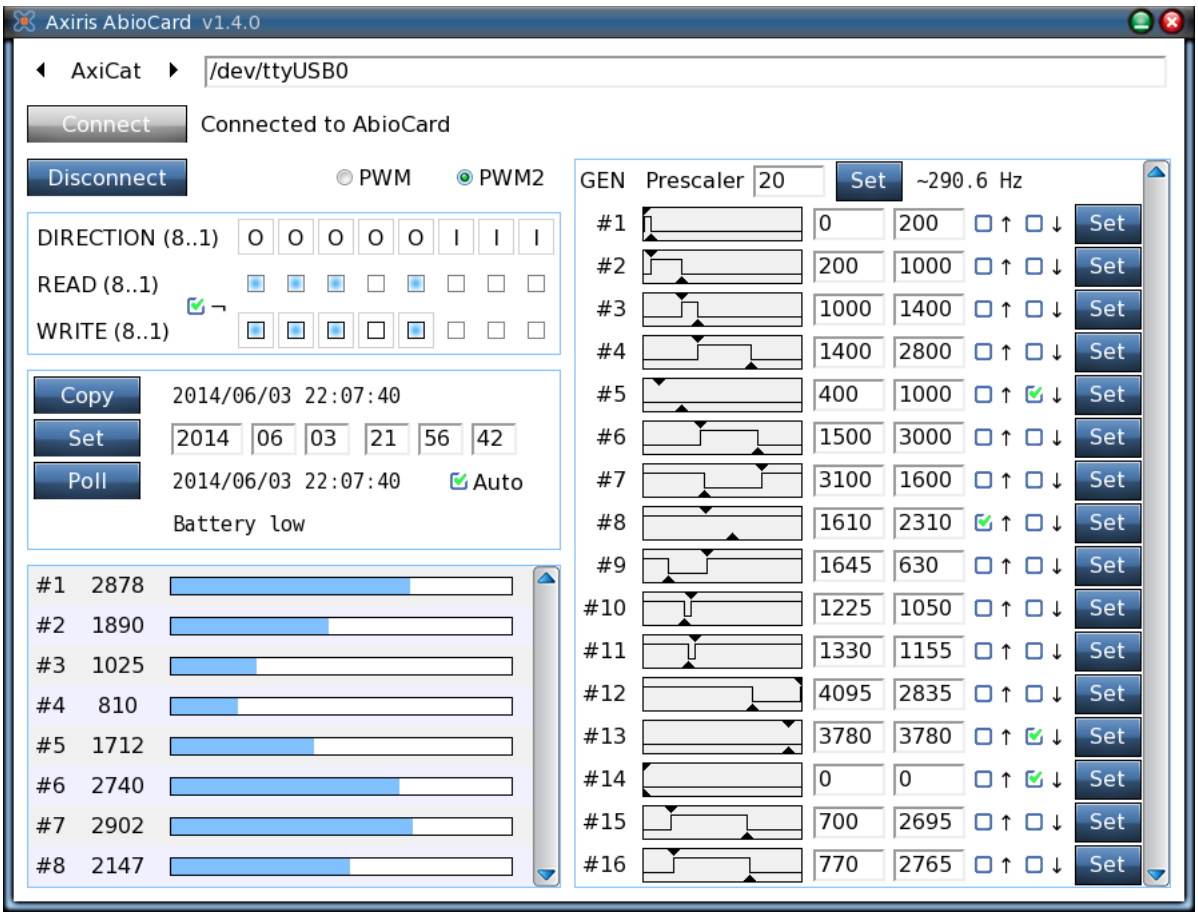
Connecting an AbioCard to any Supported Computer

With the help of the AxiCat, you can connect an AbioCard model A or B to any computer that's capable of running the AbioCard software.

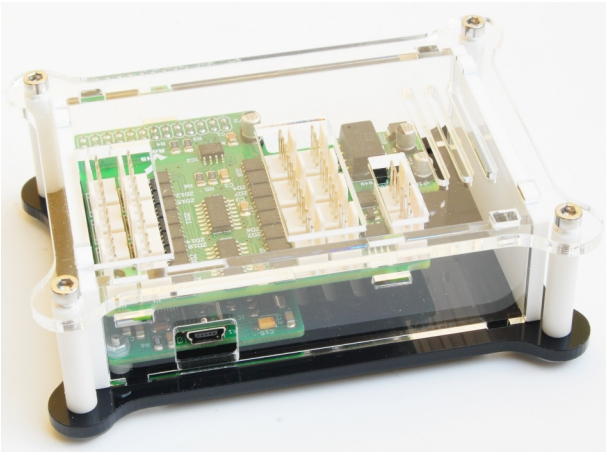


Make sure the jumper (JP1) is set to 3.3 V operation as shown in the picture above!

The following is an example of the **abiocardgui** program running in Linux and talking with an AbioCard model B mounted on an AxiCat. Linux has assigned serial path /dev/ttyUSB0 to the board.



Connecting an AbioWire to any Supported Computer



Thanks to the AxiCat, you can attach an AbioWire board to any computer that's capable of running the 1-Wire Automation Server software.

A dedicated enclosure for the AxiCat with AbioWire combo is available. The acrylic enclosure allows you to set up the hardware for ease of use.

The AbioWire is a 3.3 V device. Make sure the jumper (JP1) on the AxiCat is set to 3.3 V operation!

Suppose your AxiCat with AbioWire combo is plugged into a computer running Windows and the AxiCat is enumerated as serial port COM4. Feed the following commands to the 1-Wire Automation Server in order to use the AbioWire:

```
adapter "aw" add serial "\\.\COM4" axicat twi abiowire
adapter "aw" enable
```

Execute the **dump dn** command to query the device node structure for the enabled adapter:

```
dump dn
Device Nodes:
+ ROOT
  + SERIAL: "\\.\COM4" [open]
    + AXICAT [open:1]
      + I2CBUS: twi [open:3]
        + OWS-DRV: DS2482 ad 18h [open] -> adapter "aw1"
        + OWS-DRV: DS2482 ad 19h [open] -> adapter "aw1"
        + OWS-DRV: DS2482 ad 1Ah [open] -> adapter "aw1"
```

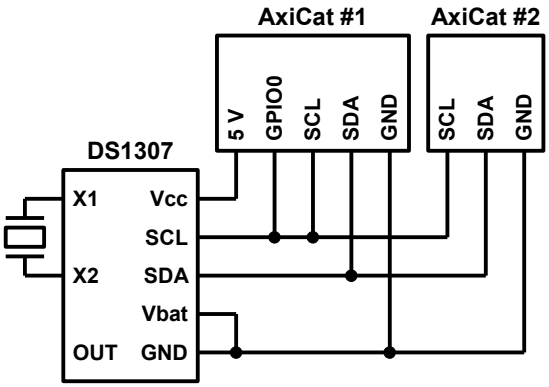
Likewise you can add multiple AbioWire adapters to the 1-Wire Automation Server. Suppose you connect two combos to a computer with Linux and the serial ports are enumerated as ttyUSB0 and ttyUSB1. The following commands will add and enable the AbioWire adapters in the 1-Wire Automation Server:

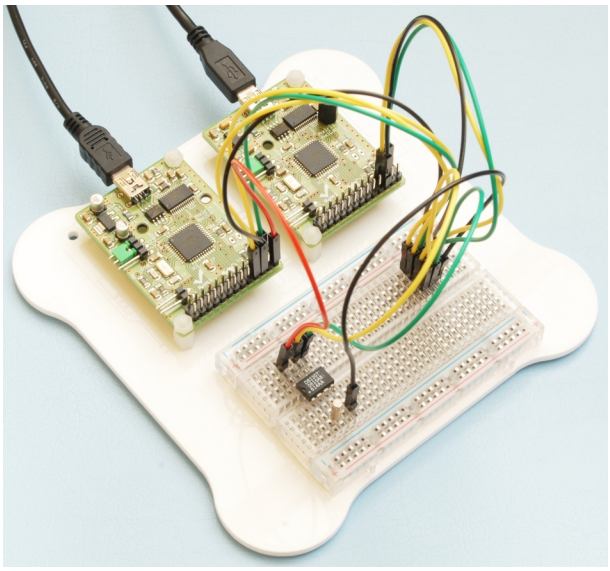
```
adapter "aw1" add serial "/dev/ttyUSB0" axicat twi abiowire
adapter "aw2" add serial "/dev/ttyUSB1" axicat twi abiowire
adapter "aw1" enable
adapter "aw2" enable
```

I2C Multi-Master Arbitration Test Setup

Multi-master arbitration is a powerful feature of the I2C bus. It allows multiple masters to negotiate access to a single I2C bus. It's an interesting subject to take a closer look at.

The hard part of multi-master arbitration is that it may be hard to reproduce with the tools you have at hand. We've devised a test setup that allows you to easily recreate various arbitration situations. The setup uses two AxiCat boards, a DS1307 slave, a breadboard and a bunch of jumper wires.





The two AxiCat devices act as I2C masters. They'll try to access the DS1307 slave device while honoring the rules of arbitration.

GPIO0 does the magic. Its function is to hold low SCL to prevent the I2C masters from accessing the bus. This state is also known as clock stretching. While SCL is held low, the masters are instructed to start a transfer. They won't access the bus though. When GPIO0 releases SCL, both masters will access the bus at the same time.

GPIO0 must be used in a specific way. First, initialize to logic zero. Next, manipulate the direction to control the SCL line. When GPIO0 is set as output, it holds SCL low. When GPIO0 is set as input, the pin becomes tri-state thus releasing SCL. Remember that SDA and SCL are pulled upon the board, hence the state of a released SCL is logic one.

Now run an instance of program **axicatcl** with each AxiCat and enter the following commands.

AxiCat #1:

Command	Protocol bytes		Description
`io 0 0`	`Tx`	`05h 00h`	Set GPIO0 to logic zero
`io 0 out`	`Tx`	`04h 80h`	Set GPIO0 to output: SCL held low
`twi ena`	`Tx`	`22h`	Enable TWI
`twi start 68h w`	`Tx`	`24h D0h`	SLA+W address 68h
`twi mtx 0 0 0 0`	`Tx`	`26h 03h` `00h 00h 00h 00h`	Transmit 4 data bytes
`twi stop`	`Tx`	`25h`	STOP
`rx`	`Rx`		No data due to SCL held low

AxiCat #2:

Command	Protocol bytes		Description
`twi ena`	`Tx`	`22h`	Enable TWI
`twi start 68h w`	`Tx`	`24h D0h`	SLA+W address 68h
`twi mtx 0 1 0 0`	`Tx`	`26h 03h` `00h 01h 00h 00h`	Transmit 4 data bytes
`twi stop`	`Tx`	`25h`	STOP
`rx`	`Rx`		No data due to SCL held low

Now we're going to release the SCL line and see how the I2C master will proceed.

AxiCat #1:

Command	Protocol bytes		Description
`io 0 in`	`Tx`	`04h 00h`	Set GPIO0 to input: release SCL
`rx`	`Rx`	`24h D0h`	SLA+W address 68h received ACK response
		`26h 03h 03h`	Transmitted all 4 data bytes
		`25h`	STOP signal generated

AxiCat #2:

Command	Protocol bytes		Description
`rx`	`Rx`	`24h D0h`	SLA+W address 68h received ACK response
		`23h`	Arbitration lost during following command
		`26h 03h 01h`	Transmitted 2 of 4 data bytes
		`A5h`	Skipped STOP

The results indicate that AxiCat #1 has won the arbitration during the transmission of the second data byte.

Once you get the hang of it, you can try out other combinations of SLA+W, SLA+R, slave address and data bytes to see how arbitration behaves.

6 Serial Paths

You need to specify a serial path in order to communicate with a serial port. The next sections explain in more detail how you specify serial paths in each supported operating system.

Linux

Serial ports are accessible in the device directory structure. A serial path starts with **/dev**. A serial path is case-sensitive.

The following table summarizes serial paths that are commonly found on Linux systems:

Serial Path	Serial Port
/dev/ttyS0	The computer's 1st on-board serial port
/dev/ttyS1	The computer's 2nd on-board serial port
/dev/ttyS2	The computer's 3rd on-board serial port
/dev/ttyS3	The computer's 4th on-board serial port
/dev/ttyUSB0	1st USB serial adapter
/dev/ttyUSB1	2nd USB serial adapter
/dev/serial/by-id/	This directory contains symbolic links to serial devices. Each symbolic link name identifies a specific device.
/dev/serial/by-path/	This directory contains symbolic links to serial devices. Each symbolic link name represents a hardware path to the device, like a specific USB port on your computer.

Here are some useful commands you can run to get information about present serial devices and their corresponding serial path. The following commands were run on a Linux system with one on-board UART and one connected USB serial adapter.

Filter information from the kernel message buffer:

```
$ dmesg | grep 'tty'
[    0.000000] console [tty0] enabled
[    0.516785] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
[    0.517463] 00:0b: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
[    0.559097] tty tty55: hash matches
[   66.076326] usb 2-1: FTDI USB Serial Device converter now attached to ttyUSB0
```

Use the **setserial** command to produce a list of serial paths:

```
$ setserial -g /dev/ttyS* /dev/ttyUSB*
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
/dev/ttyS1, UART: unknown, Port: 0x02f8, IRQ: 3
/dev/ttyS2, UART: unknown, Port: 0x03e8, IRQ: 4
/dev/ttyS3, UART: unknown, Port: 0x02e8, IRQ: 3
/dev/ttyUSB0, UART: unknown, Port: 0x0000, IRQ: 0, Flags: low_latency
```

List serial devices in the device directory:

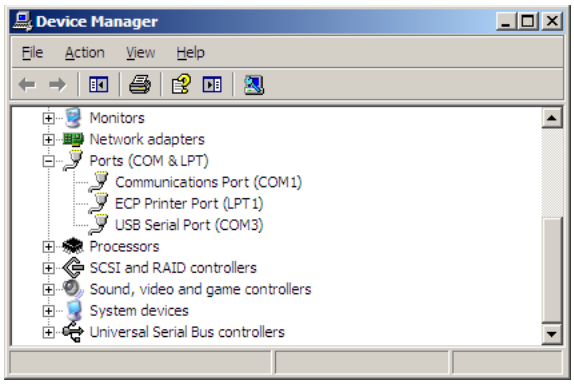
```
$ ls -l /dev/ttyS* /dev/ttyUSB*
crw-rw---- 1 root dialout   4, 64 2012-02-22 14:19 /dev/ttyS0
crw-rw---- 1 root dialout   4, 65 2012-02-22 14:19 /dev/ttyS1
crw-rw---- 1 root dialout   4, 66 2012-02-22 14:19 /dev/ttyS2
crw-rw---- 1 root dialout   4, 67 2012-02-22 14:19 /dev/ttyS3
crw-rw---- 1 root dialout 188,  0 2012-02-22 14:20 /dev/ttyUSB0
```

Windows

Serial ports are accessible through the Win32 device namespace, which is part of the NT namespace. As such a serial path starts with **\\.** followed by the device name of the serial port. A serial path is case-insensitive.

A serial port is typically named **COM<x>** where **<x>** is a number between 1 and 256. Other naming schemes may apply and aliases may exist, depending on the serial driver that controls the serial port.

You can obtain a list of available serial ports in the device manager. Open the device manager and view devices by type. The section named *Ports (COM & LPT)* contains all available serial and parallel ports.



The device name of a serial port is shown between parentheses. In the picture to the right, you can see two serial ports. COM1 is the PC's on-board serial port, COM3 represents a USB serial adapter.

The serial paths to the serial ports in the picture are:

Serial Path	Serial Port
\\.\COM1	*Communications Port (COM1)*
\\.\COM3	*USB Serial Port (COM3)*

Serial paths like **COM1** (that's without the **\\.** prefix) will work because COM1 to COM9 are reserved names in the NT namespace. COM10 to COM256 aren't reserved names and you'll have to specify the **\\.** prefix with these device names. By comparison, serial path **\\.\COM100** will work but serial path **COM100** won't work.

7 Legal Information

Disclaimer

Axiris products are not designed, authorized or warranted to be suitable for use in space, nautical, space, military, medical, life-critical or safety-critical devices or equipment.

Axiris products are not designed, authorized or warranted to be suitable for use in applications where failure or malfunction of an Axiris product can result in personal injury, death, property damage or environmental damage.

Axiris accepts no liability for inclusion or use of Axiris products in such applications and such inclusion or use is at the customer's own risk. Should the customer use Axiris products for such application, the customer shall indemnify and hold Axiris harmless against all claims and damages.

Trademarks

"Atmel" is a registered trademark or trademark of Atmel Corporation or its subsidiaries, in the US and/or other countries.

"Microchip" is a registered trademark of Microchip Technology Incorporated in the U.S.A. and other countries.

All product names, brands, and trademarks mentioned in this document are the property of their respective owners.

8 Contact Information

Official website: http://www.axiris.eu/